СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc40641068)

[1. ОПИСАТЕЛЬНЫЙ РАЗДЕЛ 5](#_Toc40641069)

[1.1 Описание предметной области 5](#_Toc40641070)

[1.2 Описание выбранного жанра 5](#_Toc40641071)

[1.3 Этапы разработки 6](#_Toc40641072)

[1.4 Описание модулей среды разработки 10](#_Toc40641073)

[1.4 Разработка компилятора команд 14](#_Toc40641074)

[1.5 Разработка меню игры 16](#_Toc40641075)

[1.6 Разработка программной логики 17](#_Toc40641076)

[1.7. Описание уровней игры 26](#_Toc40641077)

[2. ТЕХНОЛОГИЧЕСКИЙ РАЗДЕЛ 30](#_Toc40641078)

[2.1 Обзор аппаратных средств 30](#_Toc40641079)

[2.2 Обзор программных средств 30](#_Toc40641080)

[2.3 Описание ПО для функционирования игры 30](#_Toc40641081)

[2.4 Руководство пользователя 32](#_Toc40641082)

[3. ОХРАНА ТРУДА 33](#_Toc40641083)

[3.1. Сведения об опасных и вредных факторах 33](#_Toc40641084)

[3.2. Производственная санитария 34](#_Toc40641085)

[4. ЭКСПЕРИМЕНТАЛЬНЫЙ РАЗДЕЛ 35](#_Toc40641086)

[4.1. Тестирование игры 35](#_Toc40641087)

[ГРАФИЧЕСКИЙ РАЗДЕЛ 37](#_Toc40641088)

[ЗАКЛЮЧЕНИЕ 43](#_Toc40641089)

[ПРИЛОЖЕНИЯ 46](#_Toc40641090)

# ВВЕДЕНИЕ

Ежедневно по всему миру создаются различные видеоигры. Цель каждой игры – рассказать свою собственную, уникальную, неповторимую историю. Игрок погружается в чуждый для него мир, постепенно выясняя пока незнакомые ему правила, сосредоточенно и требовательно исследует он мир бескрайных виртуальных горизонтов. Для каждого творца, для каждого, кто хотел бы воплотить свои мысли в реальность, проявить мельком промелькнувший образ в голове – для тех созданы игры. Если ранее, в незапамятные времена, наши предки были ограничены возможностями создания различных сцен – то теперь творцы поистине безграничны. Доступны слух, зрение, а в наиболее современных играх осязание и обоняние в игре. Это поистине другой мир, это поистине незабываемое приключение.

Целью игрового проекта ставится разработка обучающей игры с элементами космической аркады.

Задачи***:***

1. изучение особенностей предметной области;
2. анализ подходов и методов разработки в Unity;
3. проектирование игрового проекта;
4. реализация игрового процесса;
5. тестирование игрового проекта.

Практическая значимость заключается в освоении навыков игровой разработки: отрисовка моделей, программирование скриптов, геймлевел-менеджмент, композиция и взаимодействие игровых объектов.

Следует отметить, что на сегодняшний день геймдев-разработка является одним из самых приоритетных направлений в программировании: это целая отдельная область, где есть свои подходы, монетизация, нюансы и сообщество. Разработка игр перестала зависеть от крупных издателей: среды разработки стали настолько удобными, что теперь для разработки не требуются специфические навыки. Безусловно, тот факт, что многие из этих сред бесплатны для использования – еще одно значительное преимущество отдельного разработчика перед плеядой крупных издательств.

# 1. ОПИСАТЕЛЬНЫЙ РАЗДЕЛ

## 1.1 Описание предметной области

Разработка обучающей игры – разработка не сколько игры, сколько интерактивного учебника, предназначенного для обучения игрока. Примеров данного вида игр достаточно мало: каждая новая в этой области носит самобытный характер, и, таким образом, нету какого-либо единого свода правил для данного жанра видеоигр. Следует отметить, что все игры должны быть и развлекающими, и интересными, и создаваемая видеоигра не должна стать исключением. Совместить скучный процесс изучения нового с погружением в игровой мир – это увлекательный и многоступенчатый процесс разработки.

Разработчик не ставит цель создать проект мирового уровня – преследуется цель создание такой игры, которая сможет действительно дать реальные навыки реальному пользователю. В угоду популярности ставится функциональность и доступность понимания.

Создаваемая игра не должна обладать повышенными требованиями к производительности и должна запускаться на любом устройстве независимо от разрешения, мощностей и прочих характеристик устройства. В качестве требований предъявляется только наличие привычной периферии: монитор, клавиатура, компьютерная мышь.

## 1.2 Описание выбранного жанра

Строго говоря, созданная игра не относится ни к одному из известных жанров, так как представляет собой скорее интерактивный учебник, чем игру. Меж тем, можно найти прообраз аркады: герой путешествует между уровнями, взаимодействуя с интерактивными объектами.

Аркадные игры часто имеют короткие уровни, простые и интуитивно понятные схемы управления, а также быстро возрастающую сложность. Игры на консолях или ПК можно назвать "аркадными играми", если они разделяют эти качества. Многие независимые разработчики сейчас выпускают игры в аркадном жанре, которые разработаны специально для использования в Интернете. Эти игры обычно разрабатываются с использованием Flash / Java/ DHTML и запускаются непосредственно в веб-браузерах. Аркадные гоночные игры имеют упрощенный физический движок и не требуют много времени на обучение по сравнению с гоночными симуляторами. Автомобили могут резко поворачивать без торможения или недостаточной поворачиваемости, а соперники ИИ иногда запрограммированы так, что они всегда находятся рядом с игроком (эффект резиновой ленты).

В прошлом в аркадных видеоиграх с монетным управлением обычно использовалось индивидуальное игровое оборудование, часто с несколькими процессорами, узкоспециализированными звуковыми и графическими чипами и новейшими дорогими технологиями отображения компьютерной графики. Это позволило аркадным системным платам производить более сложную графику и звук, чем то, что было тогда возможно на игровых консолях или персональных компьютерах, чего больше не было в 2010-х гг. Аркадное игровое оборудование в 2010-х годах часто основано на модифицированном оборудовании игровых консолей или высококачественных компонентах ПК. Аркадные игры часто имеют более захватывающее и реалистичное игровое управление, чем компьютерные или консольные игры, включая специализированные аксессуары для атмосферы или управления.

В настоящее время с появлением виртуальной реальности производители аркад начали экспериментировать с технологией виртуальной реальности.

## 1.3 Этапы разработки

**1.3.1 Подготовка к производству**

На первом этапе разработки создается дизайн-документ, в котором детально описана разрабатываемая компьютерная игра. Сюда входят все необходимые материалы для иллюстрации проектных решений. Цель документа максимально точно описать игровой процесс, графику, сюжет и так далее.

**1.3.2 Дизайн-документ**

**Вступление**

Разработчик представляет дизайн-документ компьютерной игры в жанре FPS с элементами Arcade под рабочим названием «Space C#».

Игровой проект относится к малым программным продуктам, имеет средний цикл разработки и подразумевает изучение команд языка C#.

Игрок должен управлять игровым персонажем, представленным в виде инопланетянина на космическом корабле (космическая кастрюля). Управление происходит с помощью команд на языке C#. Цель игрока – перейти на следующий уровень на каждом уровне. В процессе игрок сталкивается с препятствиями, которые он должен облетать. Неземные потусторонние фантасмагорические космические силы будут мешать ему, однако с помощью игрока персонаж сможет преодолеть все невзгоды и обрести артефакт древних.

**Игровой дизайн**

Игра представлена для одного игрока. Игровой дизайн выдержан в стиле космической тематики. Каждый уровень предоставляет возможность игроку узнать что-то новое как о изучаемом языке, так и о сюжете игры. В конце игры игрок узнает, зачем он путешествовал и зачем летел к концу путешествия.

**Условия разработки**

Команда состоит из одного человека. Создание игры «Space C#» планируется завершить в течении 1 года распределив время между такими этапами как: дизайн, программирование, графика, анимации и тестирование.

**Средство для реализации**

Для разработки игровой обучающей программы была выбрана среда разработки Unity. Цель игровой программы – подготовить начинающих программистов организации к реальным задачам посредством изучения языка C#. При прохождении игры пользователь должен печатать команды, дабы передвигать игрового персонажа на следующий уровень. По мере прохождения игры команды усложняются.

Unity – это, проще говоря, самый популярный игровой движок в мире. Он объединяет в себе массу функций и достаточно гибок, чтобы сделать практически любую игру, которую вы можете себе представить.

Обладая непревзойденными кросс-платформенными функциями, Unity пользуется популярностью как у разработчиков инди, так и у студий AAA формата. Ее использовали, чтобы создавать игры, как Покемон GO, Heathstone, Cuphead, и многое другое.

В то время как 3D в названии, Unity 3D также предоставляет инструменты для разработки 2D-игр.

Программисты любят его из-за скриптового API C# и встроенной интеграции с Visual Studio. Unity также предлагает JavaScript в качестве языка сценариев и MonoDevelop в качестве IDE для тех, кто хочет альтернативу Visual Studio.

Но художники тоже любят его, так как он поставляется с мощными анимационными инструментами, которые позволяют легко создавать свои собственные 3D-ролики или создавать 2D-анимации с нуля. Почти все может быть оживлено в едином формате.

Кроме того, Unity 3D предлагает бесплатную версию, так что разработчики могут выпускать игры, сделанные с Unity Personal, не платя за программное обеспечение, если они зарабатывают менее 100 000 долларов от игр, сделанных с Unity.

Для тех, кто готов платить, Unity предлагает некоторые дополнительные функции и гибкий план лицензирования в рамках многоуровневой модели подписки. Премиум-пользователи также получат доступ к исходному коду Unity и поддержке разработчиков.

Поскольку Unity существует с 2005 года, она создала огромное количество пользователей и удивительную библиотеку ресурсов. Мало того, что Unity имеет фантастическую документацию, но и само богатство видео и учебных пособий в интернете ошеломляет.

Unity 3D поставляется с профессиональными инструментами как для программистов, так и для художников.

Unity предоставляет рабочее пространство, которое сочетает в себе удобные для художника инструменты с управляемым компонентами дизайном, что делает разработку игр чертовски интуитивно понятной.

Как 2D, так и 3D-разработка возможна в Unity, причем 2D-физика управляется популярным движком Box2D.

Unity использует компонентный подход к разработке игр, вращающийся вокруг префабов. С помощью сборных модулей геймдизайнеры могут создавать объекты и среды более эффективно и быстрее масштабироваться.

Благодаря мощным шейдерам, материалам на основе физики, постобработке и системам освещения с высоким разрешением Unity может обеспечить впечатляющую графику по всем направлениям.

Сборная система позволяет легко повторно использовать код и ресурсы из других проектов и редактировать их для новых целей.

Построение сложных миров в Unity 3D становится вопросом сборки многих компонентов, которые состоят из их собственных компонентов.

Но главная причина выбора Unity – это огромная библиотека ресурсов, доступная каждому. Даже опытные разработчики могут сэкономить время и многому научиться у сообщества.

Unity также предлагает надежный набор облачных инструментов для легкой монетизации вашей игры и добавления многопользовательских возможностей.

Благодаря Unity Analytics, Unity Ads, Unity Collaborate и Unity Multiplayer пользователи получают доступ к невероятно совместному набору инструментов для создания динамических игр в одном месте. Очень немногие другие игровые движки предлагают такую большую централизацию.

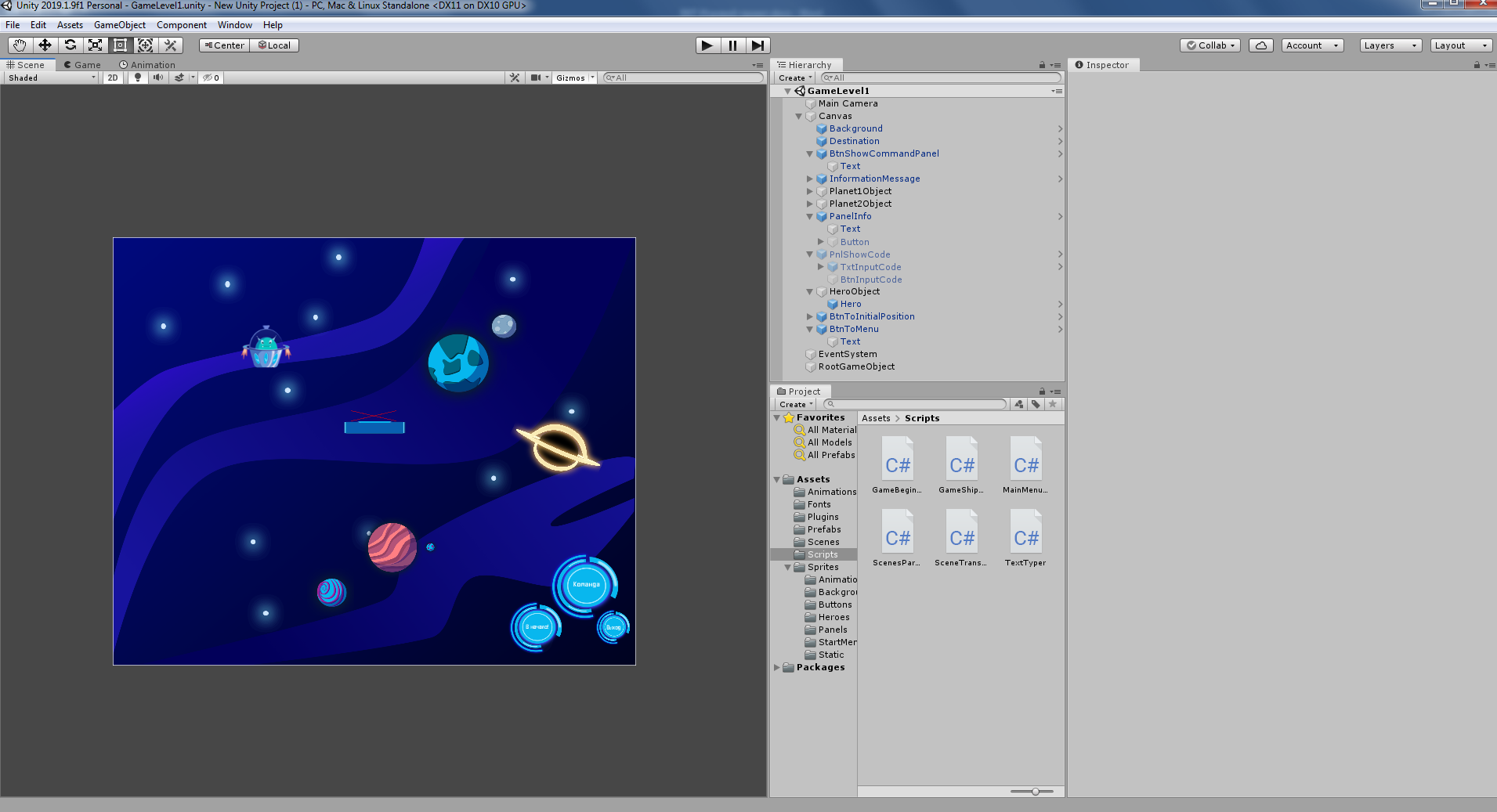


Рисунок 1 - среда разработки Unity3D

## 1.4 Описание модулей среды разработки

В Unity содержится множество возможностей, однако мы будем применять только следующие:

* взаимодействие игровых объектов друг с другом;
* взаимодействие игровых объектов с игроком;
* состояние игровых объектов с течением времени;
* логика игровых объектов и их поведение в различных ситуациях;
* представление игровых объектов на экране.

Каждый игровой объект в Unity представляет собой комплексную модель, у которой есть отдельные встраиваемые модули. Модули можно как добавлять, так и удалять, так и редактировать параметры модулей. В данной работе используются следующие модули:

**трансформация объекта**: ответственен за представление игрового объекта в пространстве, то есть содержит в себе ширину, высоту, масштаб, поворот и т.д.



Рисунок 2 - модуль трансформации

**анимация объекта**: ответственна за анимацию, в которую можно включить перемещение объекта, смена параметров объекта (к примеру, смена цвета), включение событий в определенный момент времени и т.д.

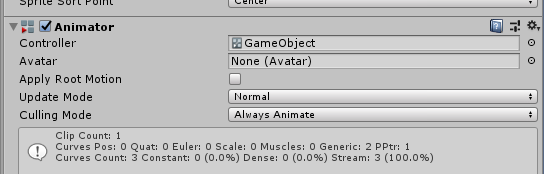


Рисунок 3 - модуль анимации

**рендерер**: в контексте 2D игры отображает игровой спрайт, а также определяет позицию спрайта на игровом слое, его цвет, тип отрисовки, материал отрисовки, ссылку на исходное изображение спрайта и т.д.

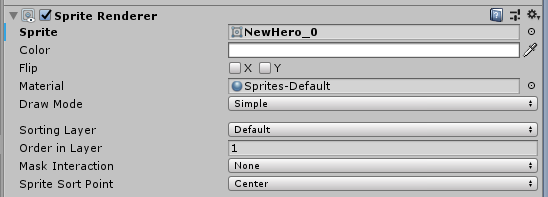


Рисунок 4 - модуль рендерера

**скрипт**: определяет игровую логикуобъекта:

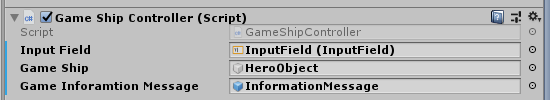


Рисунок 5 - модуль скрипта

**твердое тело**: обозначает, что данный игровой объект имеет свою массу и ведет себя как физический объект.

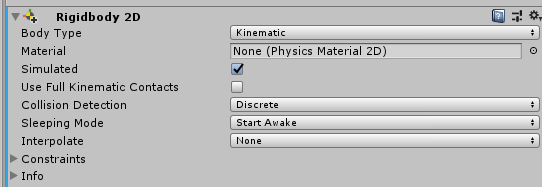


Рисунок 6 - модуль твердого тела

**коллайдер**: обозначает, что данный игровой объект может сталкиваться с другими игровыми объектами. Бывает нескольких видов, в игре используется квадрат и круг.

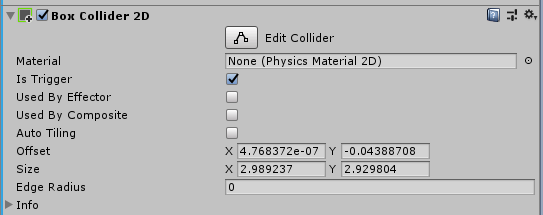


Рисунок 7 - модуль коллайдера

**изображение**: статичное игровое изображение.

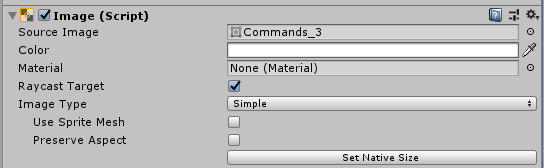


Рисунок 8 - модуль изображения

**кнопка**: представляет собой кнопку, которая может выполнять от нуля до неограниченного количества действий при нажатии. Имеет свой собственный аниматор, что означает, что доступна анимация при разных состояниях: при наведении, нажатии, по умолчанию, при фокусе и т.д.

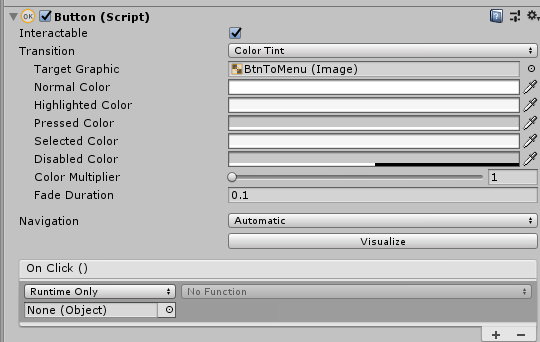


Рисунок 9 - модуль кнопки

**холст**: представляет собой область, где возможно размещение игровых объектов и их интерактивное взаимодействие. Содержит в себе несколько модулей, которые не могут работать по отдельности.

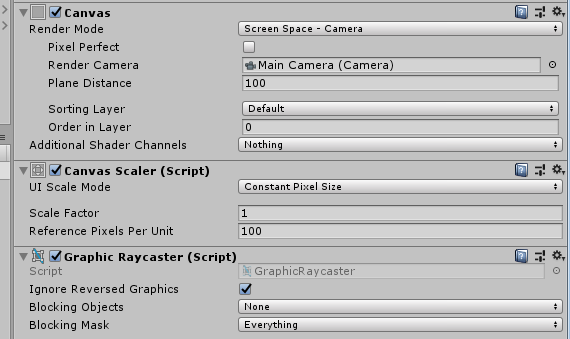


Рисунок 10 - модуль холста

камера: игровая камера, проецирующая игровое пространство на выбранную область.

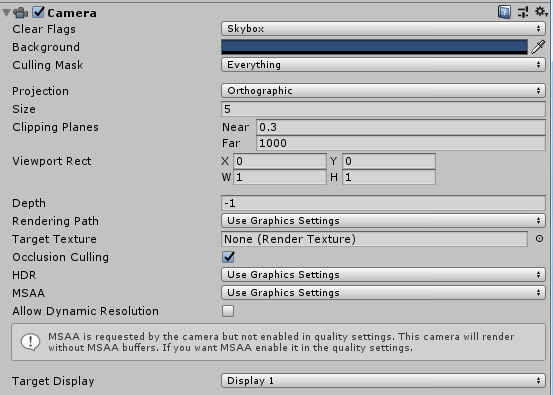


Рисунок 11 - модуль камеры

Каждый из данных модулей может быть добавлен к любому игровому объекту. В случае несовместимости модулей среда выдаст соответствующее сообщение и отменит добавление несовместимого модуля.

Отдельно рассмотрим модуль анимации: он прдеставлен в виде отдельного окна, где можно задать состояния анимации, граф переходов, условия перехода.

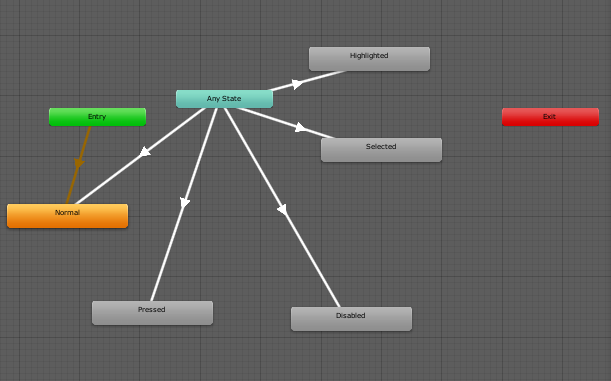


Рисунок 12 - модуль аниматор

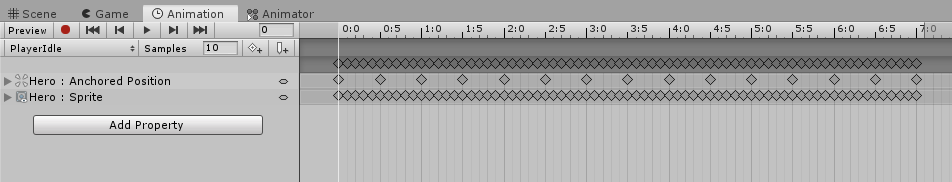


Рисунок 13 - модуль анимации

## 1.4 Разработка компилятора команд

Так как игра направлена на изучение реальных команд C#, то следует разработать библиотеку, которая могла бы в режиме реального времени компилировать код C#, подаваемых ей на вход в виде текста. Структура такого проекта должна быть следующая:

* модуль компиляции;
* игровой главный персонаж;
* точка входа.

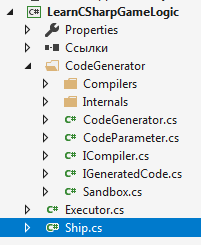


Рисунок 14 - структура проекта компилятора

Модуль компиляции не будем описывать в силу его объемности и очевидности назначения: укажем только, что компиляция кода возможна в следующем виде, где ship является параметром для указанного кода.

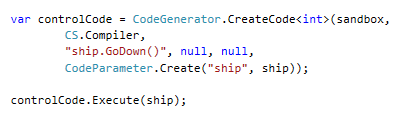


Рисунок 15 - пример компиляции в реальном времени

Как можно видеть, в проекте возможно взаимодействие только с персонажем игрока. Это ограничение вызвано с особенностью компиляции в C#: необходимо добавлять ссылки на существующие объекты, которые должны быть заранее описаны в виде отдельного класса. Так, игровой персонаж описан в виде отдельного класса Ship. Если описать другие игровые объекты – с ними также станет возможным взаимодействие.

Игровой главный персонаж имеет следующие параметры и возможности:

* передвинуть вправо/влево/вверх/вниз;
* состояние анимации (вкл/выкл);
* координаты персонажа в игровом пространстве;
* масштаб;
* послать игровое сообщение.

Если необходимо послать какую-либо специальную команду – то необходимо также определить ее в этом классе. Это не совсем правильно архитектурно: чем больше у нас будет различных спец-взаимодействий с кораблем, тем больше будет разрастаться данный класс. В дальнейшем следует категорировать все возможные виды спец-взаимодействий и распределить их по отдельным классам с архитектурным шаблоном “Команда”.

Взаимодействие игрового персонажа и его проекции в Unity осуществляется посредством событий. Это специальный инструмент в языке C#, позволяющий имплементировать в явном виде по умолчанию шаблон “Издатель-подписчик”. Игровой персонаж в библиотеке издает некие действия, а его проекция подписана на обрабатывание данных событий.

Отдельным пунктом рассмотрим точку входа. Этот класс является тем, что следует использовать скриптам из Unity для взаимодействия с библиотекой. Таким образом, мы можем валидировать и обрабатывать поступающие команды. В данном классе также определены условия, при которых поданный на вход программный код может запустить: к примеру, если игрок не ввел специальное слово в команде – то написанные им команды не будут запущены, а игра выведет соответствующее сообщение.

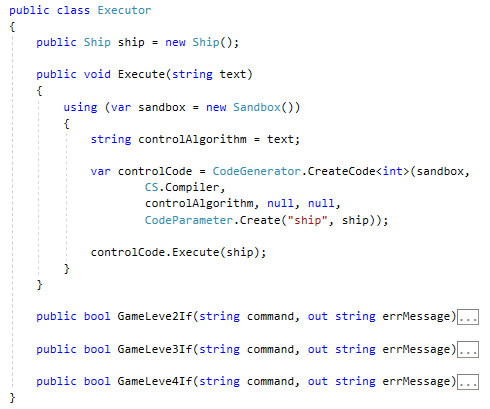


Рисунок 16 - структура класса точки входа

## 1.5 Разработка меню игры

Для меню игры было предусмотрено изначально два пункта: для старта и для выхода, так как никаких других действий не предполагалось. Как правило, меню также оснащается настройками, где можно задать привязку клавиш, уровень звука и прочие параметры. Так как описанные параметры отсутствуют в разрабатываемой игре, а весь процесс взаимодействия с игроком заранее предусмотрен- то нет необходимости делать для настроек отдельный пункт. Вместо этого был добавлен пункт информации о языке C#, который поможет игроку более хорошо понимать, с чем именно он столкнется в игре.

Итого, меню представлено в игре следующими пунктами:

* старт;
* информация о C#;
* выход.

Каждый пункт имеет свою собственную уникальную анимацию при наведении и нажатии. При нажатии на каждый пункт соответственно игрок или начнет новую игру, или откроется браузер с адресом <https://social.msdn.microsoft.com/profile>, или приложение закроется.

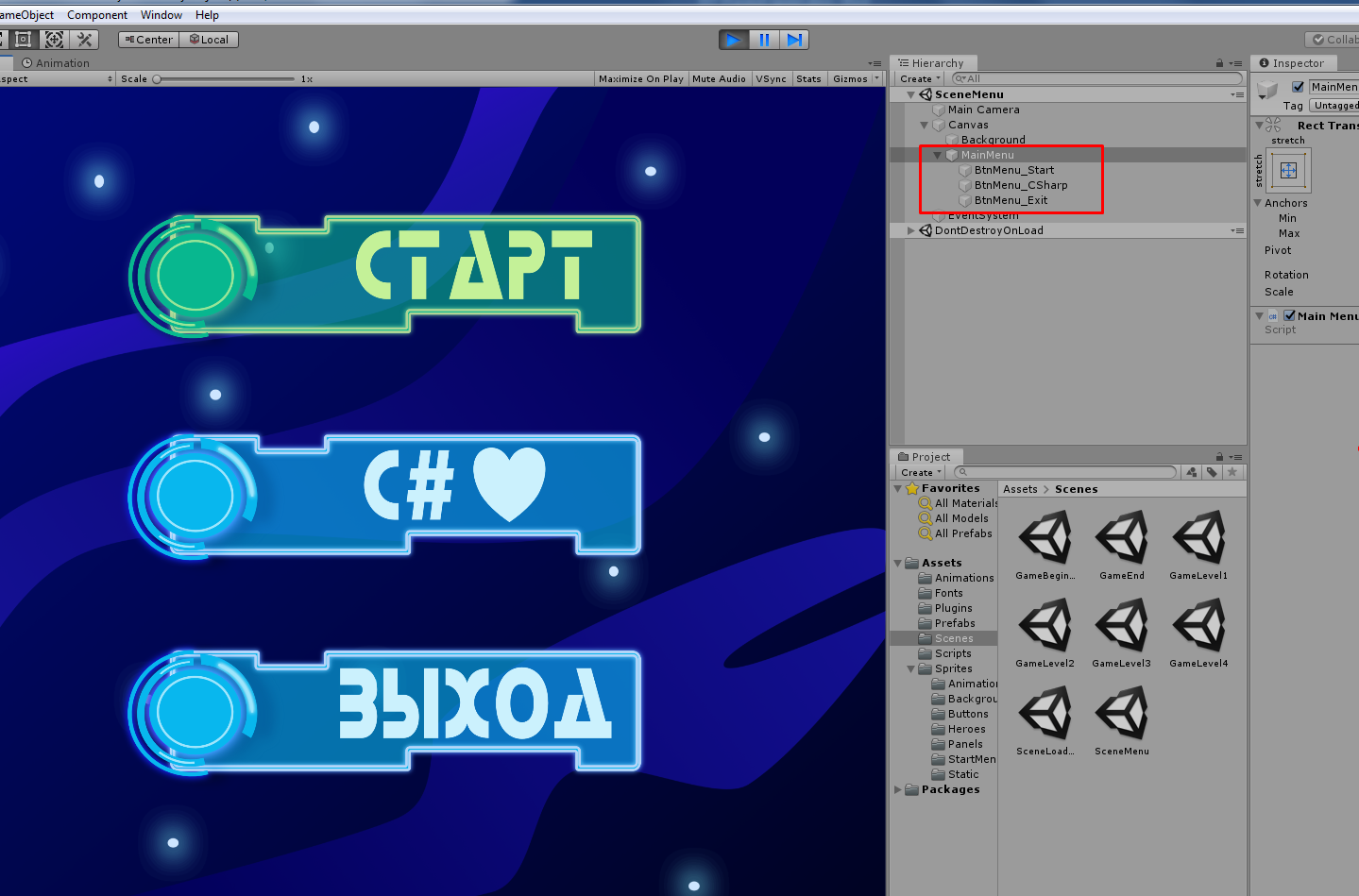


Рисунок 17 - игровое меню в реальном времени

Пункты меню доступны в виде префабов. Все пункты меню объединены под одним родителем, что позволяет эффективно взаимодействовать с ними как с единым объектом: трансформировать, перемещать и другие аналогичные действия.

## 1.6 Разработка программной логики

**Взаимодействие скрипта и среды разработки**

Игровой уровень состоит преимущественно из игровых изображений и программной логики. И если изображения добавить достаточно просто, то с логикой следует потратить больше времени.

Для написания логики используется следующий механизм: для игрового объекта создается игровой скрипт в виде программного кода, который непосредственно влияет на всю игру. Этот код может взаимодействовать с другими игровыми объектами, если ему предоставить ссылку на эти объекты. Строго говоря, программный код может взаимодействовать вообще с любым объектом – главное, чтобы у этого объекта было уникальное имя/тег, дабы его можно было найти и однозначно определить в игровом пространстве.

Пример игровой логики в виде скрипта можно увидеть в изображении ниже – там скрипт представлен в виде объекта с параметами. У скрипта есть название, скрипт взаимодействует с InputField и GameShip, притом эти объекты являются игровыми объектами, то есть они были добавлены через ссылку.

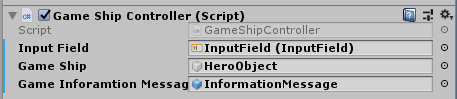


Рисунок 18 - скрипт игрового объекта

Благодаря данному скрипту становится примерно понятно его назначение и роль в системе. Однако саму работу скрипта необходимо смотреть в файле, который доступен через среду разработки (используется Visual Studio).

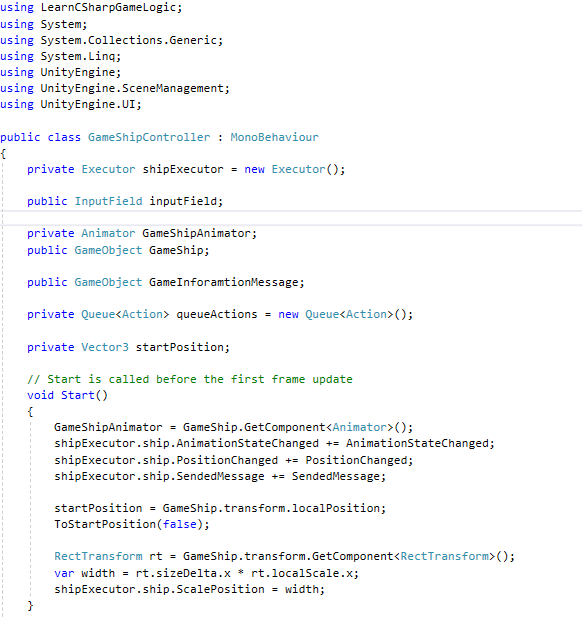


Рисунок 19 - пример игрового скрипта

**Структура игрового скрипта**

У каждого игрового скрипта есть две игровые функции по умолчанию:

* Start (действия при старте игры);
* Update (действия при каждом моменте игрового времени).

Данные действия хоть и доступны по умолчанию – их можно удалить. В большинстве случаев это имеет смысл: все действия при старте должны быть собраны в едином месте, а делать что-либо при каждом моменте игрового времени – слишком накладно в плане производительности. Следует использовать такой инструмент, как события, дабы корректно влиять на игру. В этом скрипте метод Start не удален, так как при старте игры следует инициализировать обработчики происходящих событий в игре. Также следует получить доступ к игровому объекту.

Каждый скрипт имеет следующую атрибутику:

* использованные библиотеки;
* использованные поля (могут быть оформлены как параметры игрового объекта);
* использованные методы (могут быть вызваны из других скриптов).

К примеру, если сделать поле inputField публичным – то его можно использовать в виде ссылки. Аналогичным образом были добавлены и остальные параметры игрового объекта.

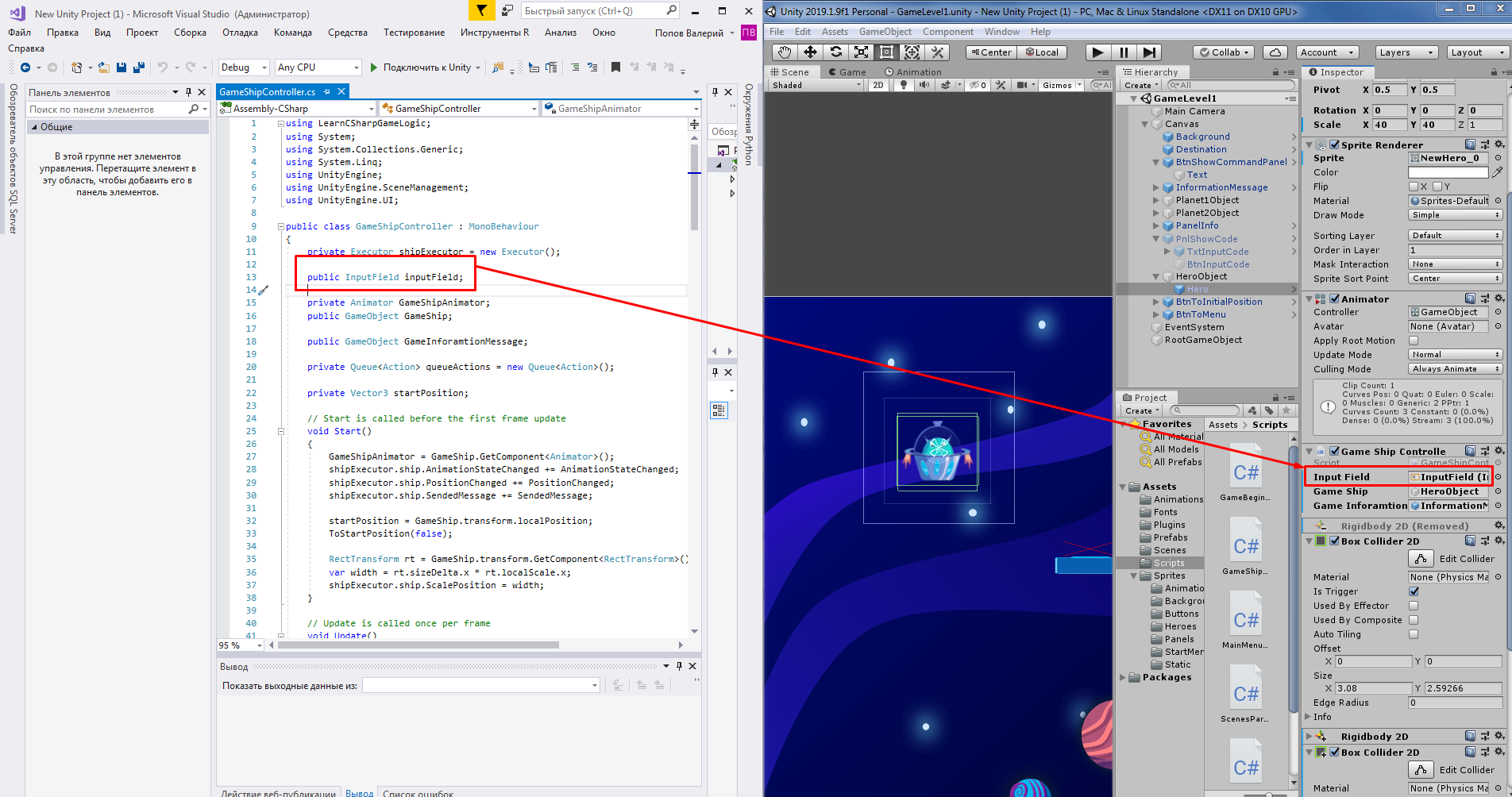


Рисунок 20 - пример использования ссылки на игровой объект

**Архитектура проектных скриптов**

При построении архитектуры программного проекта было принято решение разделить игровые скрипты на следующие категории:

* скрипт игрового персонажа;
* скрипт взаимодействия с планетами;
* скрипт взаимодействия с лунами;
* скрипты для каждого отдельного уровня (логика может различаться);
* скрипты вспомогательные (показ панели команд, игровое меню и т.д.).

Все игровые скрипты располагаются в единой сборке, в едином проекте, в отдельной папке. Как можно видеть, созданные скрипты придерживаются описанной выше категоризации:

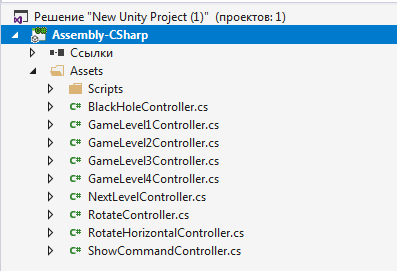


Рисунок 21 - игровые скрипты

Отметим, что, хотя названия файлов скриптов можно давать в виде кириллицы, и называть кириллицей все поля, методы – этого не стоит делать, потому что в дальнейшем, если проект будет распространен на мировом уровне, то иностранные разработчики не смогут воспринять написанный на русском языке программный код.

Каждый скрипт написан в ООП-стилистике, и наследуется от базового класса MonoBehaviour: без этого наследования скрипт не может взаимодействовать с тем игровым объектом, к которому он подсоединен.

**Рассмотрение базовых скриптов**

Рассмотрим наиболее важные скрипты, в порядке от самого простого к самому сложному.

Скрипт перехода на следующий уровень является самым простым: его функция – ждать, пока игрок не столкнется с игровым объектом, и затем загрузить следующий уровень.

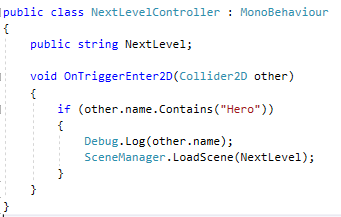


Рисунок 22 - скрипт перехода на следующий уровень

Скрипты планет и лун должны выполнять следующие функции:

* вращение объекта с задаваемой скоростью;
* реакция на столкновение с игроком;
* отображение информационной подсказки.

Соответственно, для каждой из этой функции следует создать ссылки на взаимодействующие объекты, добавить модули, определить логику отображения подсказки. Продемонстрируем скрипт лун: скрипты планет построены аналогичным образом и задействуют схожую логику. У лун только одно отличие: им необходимо указать, вокруг какого объекта следует вращаться. Таким образом, у лун есть как вращение вокруг собственной оси, так и вокруг планеты. Есть также специальный скрипт, который имитирует движение луны не вокруг, а поперек планеты.

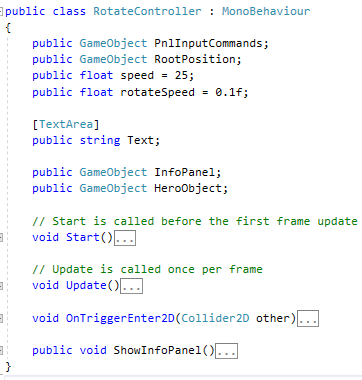


Рисунок 23 - скрипт луны

Скрипты игровых уровней построены по единообразному принципу: отображать панель команд на каждом уровне, добавляя прозрачности при этом игровому персонажу в целях удобства отображения и ввода команд.

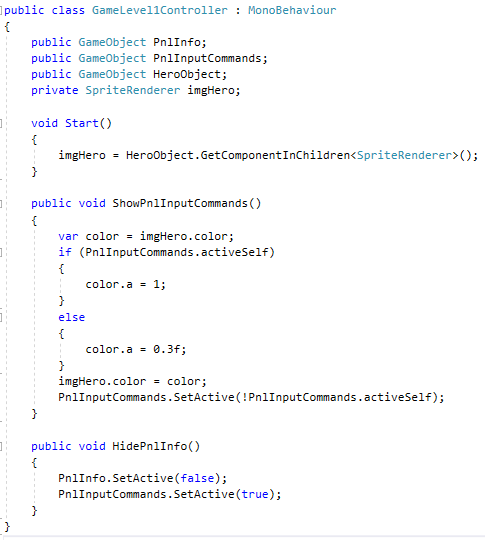


Рисунок 24 - скрипт игрового уровня

Остальные игровые скрипты являются малозначимыми, но упомянем их в качестве упоминания о том, насколько много в игре скриптов:

* скрипт отображения текста с эффектом печатной машинки;
* скрипт взаимодействия с меню;
* скрипт перехода отображения на следующий уровень;
* скрипт вступительной фазы игры;
* параметры перехода на следующий уровень (название следующего уровня).

Перейдем к наиболее объемной программной части: программирование игрового персонажа.

**Описание программной логики игрового персонажа**

Игрок должен изучать C#. Для этого необходимо предоставить игроку возможность наглядно видеть, что введенные им команды как-то взаимодействуют с игрой. Соответственно, для ввода кода необходимы следующие компоненты:

* панель ввода с текстовым встроенным редактором;
* кнопка выполнения команд;
* кнопка, скрывающая/показывающая панель ввода.

данные объекты представлены на игровом поле и выделены желтой рамкой:

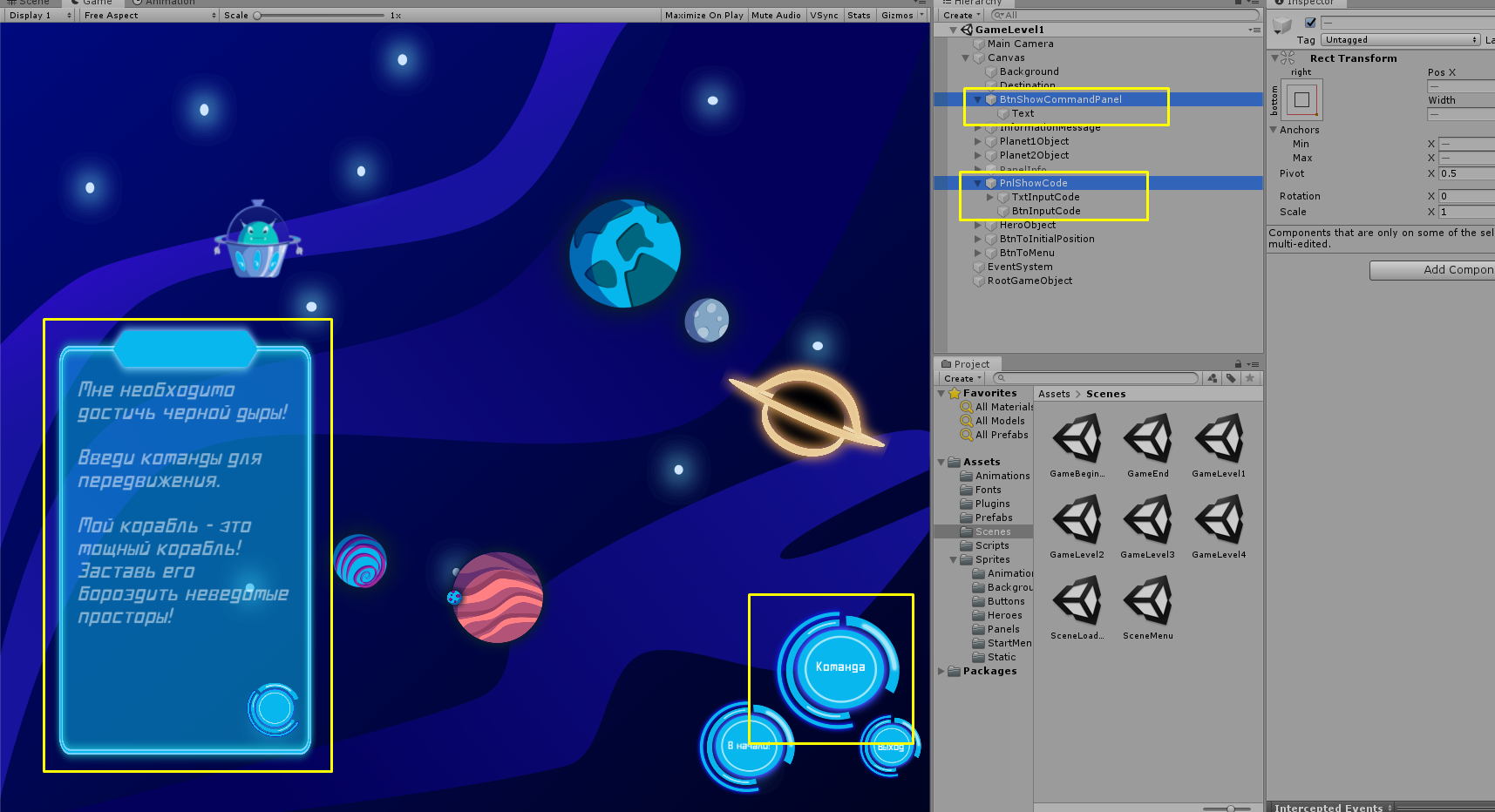


Рисунок 25 - ввод игровой команды

Для того, чтобы игра понимала, откуда брать текст вводимой команды – необходимо указать ссылку на текстовой редактор.

В процессе ввода команд игрок может захотеть вернуть игровую ситуацию в ее начальное состояние, дабы попробовать заново пройти уровень. Соответственно, для этого нужна кнопка, позволяющая это сделать.

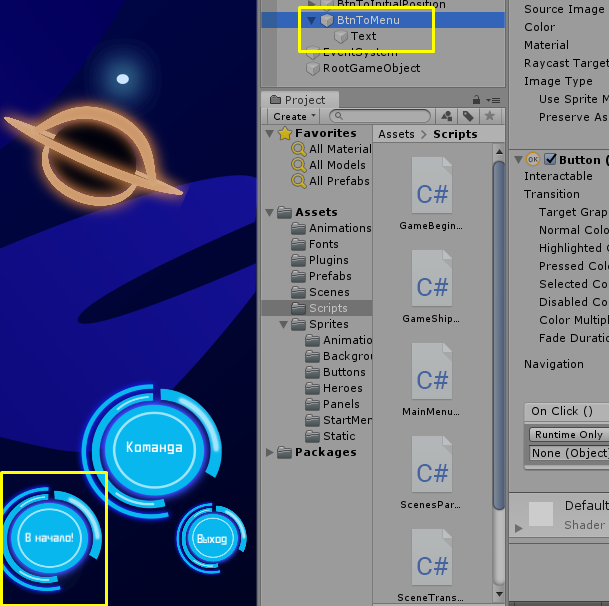


Рисунок 26 - кнопка перемещения в начало

Сам игровой персонаж – это игровой объект внутри игрового объекта. Данный трюк необходим для того, чтобы анимация игрового персонажа была относительна, а не абсолютна: это позволит космическому кораблю парить в пространстве на любой точке игрового поля.

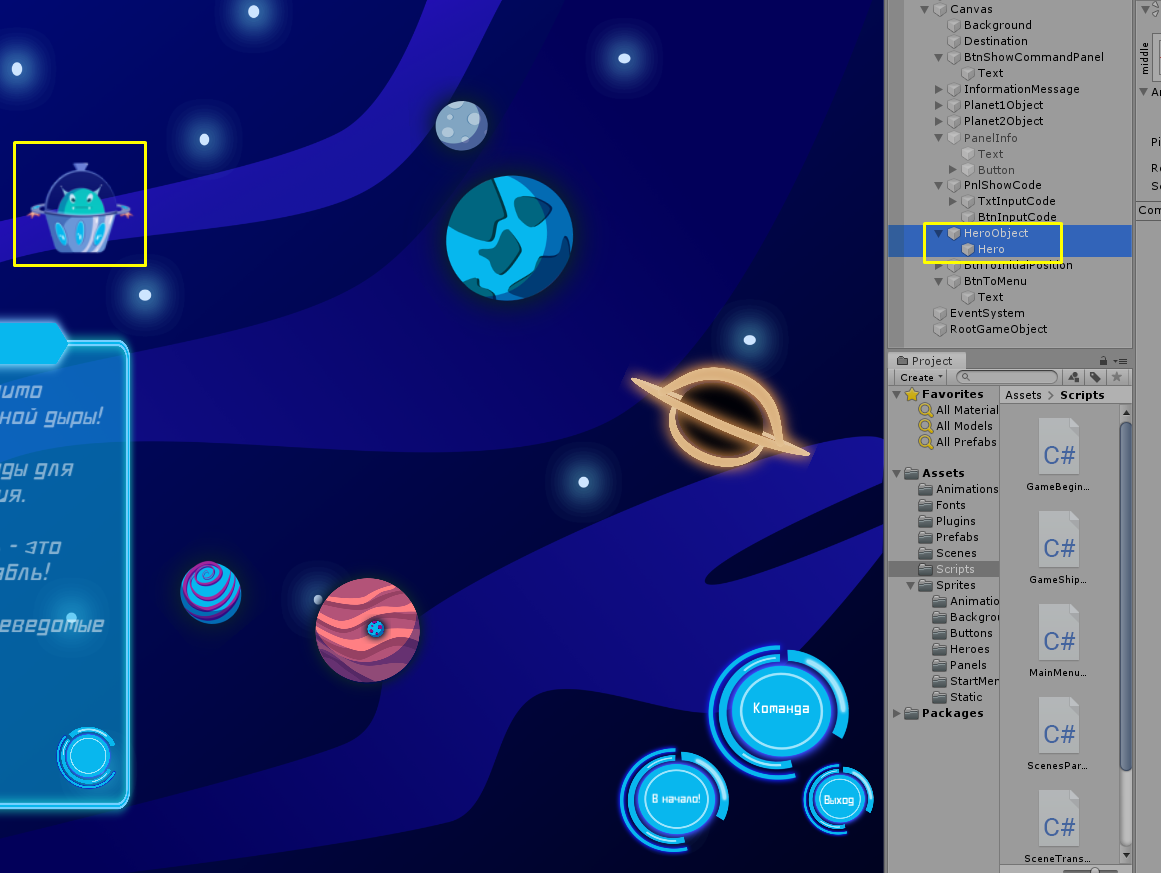


Рисунок 27 - игровой персонаж

Несмотря на то, что все эти объекты на первый взгляд из разных игровых областей – управляет ими всеми один скрипт. К его рассмотрению и перейдем. Данный скрипт назван как GameShipController, и он достаточно объемен.

Так как изучение и пошаговое рассмотрение всех задействованных алгоритмов скрипта выльется в достаточно большой объем трудно интерпретируемого текста, то представим функциональные возможности данного скрипта:

* взаимодействие с библиотекой компиляции;
* игровое поле, куда вводится текст команд;
* аниматор игрового объекта персонажа;
* игровой персонаж в виде объекта;
* игровая панель, куда следует выгружать информационное сообщение;
* очередь действий, которые следует воспроизвести;
* начальная позиция на уровне;
* действия при старте игры (кеширование и подписка на события);
* при каждом обновлении игрового кадра – воспроизводить действия из очереди действий, если они там есть;
* переместиться в начальную позицию;
* загрузить в библиотеку компиляции введенный код;
* отобразить один из заготовленных внутриигровых текстов;
* реагирование на смену позиции персонажа;
* реагирование на смену состояния анимации персонажа;
* реагирование на поступившее игровое сообщение.



Рисунок 28 - скрипт игрового персонажа

## 1.7. Описание уровней игры

Уровни игры выстроены в возрастающем порядке с увеличивающейся сложностью. Определены следующие уровни:

1. вступительный;
2. знакомство с командами;
3. знакомство с переменными;
4. изучение типов переменных;
5. использование циклов.

Каждый уровень составлен индивидуальным образом: разные информационные подсказки, игровые объекты, наполнение уровня.

Представим каждый уровень в виде изображения:



Рисунок 29 - вступительный уровень



Рисунок 30 - первый уровень



Рисунок 31 - второй уровень



Рисунок 32 - третий уровень



Рисунок 33 - четвертый уровень

Также доступен заключительный уровень игры.



Рисунок 34 - заключительный уровень

# 2. ТЕХНОЛОГИЧЕСКИЙ РАЗДЕЛ

## 2.1 Обзор аппаратных средств

Для реализации проекта необходимы аппаратные средства, представленные в таблице 1.

Таблица 1. Используемые аппаратные средства

|  |  |
| --- | --- |
| **Конфигурация** | **Использованные компоненты** |
| Процессор | не старше 2008 года выпуска, не менее 1,7 ГГц |
| Оперативная память | от 2 гБ |
| Видеокарта | не старше 2010 года выпуска, от 256 Мб |

## 2.2 Обзор программных средств

Для создания компьютерной игры понадобилось профессиональное программное обеспечение.

**Affinity Designer** – это приложение для обработки и создания изображений в векторном формате.

## 2.3 Описание ПО для функционирования игры

**DirectX**

DirectX – это программное обеспечение, разработанное корпорацией Майкрософт, которое взаимодействует с аппаратными компонентами компьютера. В частности, это набор интерфейсов прикладного программирования, или API, предназначенных для обработки задач, связанных с рендерингом 2D и 3D векторной графики, рендерингом видео и воспроизведением аудио на платформе Windows.

DX12 фокусируется на том, чтобы обеспечить резкое увеличение визуального разнообразия за счет значительного снижения накладных расходов на процессор, связанных с API.

Он конкурирует с OpenGL, еще одним графически ориентированным API-пакетом, представленным в 1992 году, который является открытым исходным кодом и находится в постоянном развитии технологическим консорциумом Khronos Group.

DirectX впервые появился в Windows 95. В то время большинство компьютерных игр работало на старой платформе DOS, что позволяло разработчикам взаимодействовать непосредственно с компонентами ПК, такими как аудиокарта, видеокарта, мышь и многое другое.

**Visual Studio Community 2017**

Интегрированная среда разработки Visual Studio – это творческая стартовая площадка, с помощью которой можно редактировать, отлаживать и создавать код, а затем публиковать приложение. Интегрированная среда разработки (IDE) – это многофункциональная программа, которая может быть использована для многих аспектов разработки программного обеспечения. Помимо стандартного редактора и отладчика, которые предоставляет большинство IDE, Visual Studio включает компиляторы, средства завершения кода, графические конструкторы и многие другие функции для облегчения процесса разработки программного обеспечения.

Visual Studio включает в себя редактор кода, поддерживающий IntelliSense (компонент завершения кода), а также рефакторинг кода. Встроенный отладчик работает как отладчик исходного уровня, так и отладчик машинного уровня. Другие встроенные инструменты включают профилировщик кода, конструктор для построения графических приложений, веб-конструктор, конструктор классов и конструктор схем баз данных. Он принимает подключаемые модули, которые расширяют функциональность практически на всех уровнях—включая добавление поддержки систем управления версиями (таких как Subversion и Git) и добавление новых наборов инструментов, таких как редакторы и визуальные дизайнеры для доменных языков или наборов инструментов для других аспектов жизненного цикла разработки программного обеспечения (например, клиент Azure DevOps: Team Explorer).

В разработке использовалась Community версия данной IDE. Отличия от других версий несущественны в контексте данного проекта, и поэтому не будет фокусировать на них внимание.

## 2.4 Руководство пользователя

Запустите игру (программа с расширением exe). Появится заставка, информирующая о движке Unity. Далее появится игровое меню с кнопками. Для старта игры необходимо нажать соответствующую кнопку.

**Системные требования**

Таблица 2. Системные требования

|  |  |  |
| --- | --- | --- |
| **Конфигурация** | **Минимальные** | **Рекомендуемые** |
| Процессор | От 2-х ядер | |
| Оперативная память | 2 Гб | От 4 Гб |
| Видеоадаптер | С 256 Мб видеопамяти | От 512 Мб видеопамяти |
| Жесткий диск | свободное место на диске 2 Гб; | |
| **Управление** | Клавиатура и мышь. | |
| **Установленное ПО** | Visual Studio Community 2017 и DirectX 11 | |

**Установка игры и запуск**

Установка игры не требуется: достаточно запустить исполняемый файл с названием игры “SharpShip”.

**Управление по умолчанию**

Все взаимодействие в игре осуществляется с помощью левой кнопки мыши и вводимых с клавиатуры команд.

# 3. ОХРАНА ТРУДА

## 3.1. Сведения об опасных и вредных факторах

К физическим факторам относятся:

* повышенное напряжение сети, которое может вызвать поражение током, способное привести к смерти;
* повышенный уровень шума на рабочем месте. Источники: разработчики, работа ПК. Повышенный уровень шума ухудшает условия труда, оказывает вредное воздействие на слуховой аппарат, умственную деятельность;
* повышенная или пониженная температура воздуха, повышенная или пониженная влажность воздуха рабочей зоны. Источники: ПК, источники искусственного освещения, люди;
* недостаток естественного света. Источники: площадь окон и их ориентация, степень чистоты стекла окон, широкий диапазон изменения естественного света в течение рабочего дня;
* недостаточная освещенность рабочей зоны. Источники: неправильный выбор средств искусственного освещения и их расположение;
* повышенный уровень электромагнитных излучений. Источник: системный блок.
* К психофизиологическим факторам относятся:
* перенапряжение анализаторов (зрительных и слуховых). Источник: дисплей ПК;
* монотонность труда. Источник: выполнение человеком однообразных движений кистей и пальцев рук при работе с клавиатурой и "мышью";
* эмоциональные перегрузки. Источник: характер трудового процесса программистов;
* физические перегрузки (статические). Источник: неправильно организованное рабочее место пользователя ПЭВМ, длительное пребывание в однообразном положении.

## 3.2. Производственная санитария

Санитарные нормы для этой категории:

* температура: 22-24°С (при температуре наружного воздуха ниже +10°С) и 23-25°С (при температуре наружного воздуха выше +10°С);
* относительная влажность: 40-60%;
* скорость движения воздуха: не более 0.1 м/с.

Для обеспечения оптимальных значений параметров микроклимата в замкнутых и небольших по объему помещениях при выполнении операторских работ необходимо использовать системы кондиционирования воздуха с индивидуальной регулировкой температуры и объема подаваемого воздуха.

Помещение для работы с визуальными дисплейными терминалами (ВДТ) должны иметь естественное и искусственное освещение в соответствии с ДБН В.2.5-28-2006 "Естественное и искусственное освещение".

Естественное освещение должно осуществляться через световые проемы и обеспечивать коэффициент естественной освещенности (КЕО) не ниже 1,2%.

Искусственное освещение в помещениях с рабочими местами, оборудованными ВДТ, должно осуществляться системой общего равномерного освещения. Величина освещенности на поверхности рабочего стола в зоне размещения документов должна составлять 200 - 500 лк. Если эти значения освещенности невозможно обеспечить системой общего освещения, допускается использовать местное освещение. При этом светильники местного освещения следует устанавливать таким образом, чтобы не создавать бликов на поверхности экрана и увеличивать освещенность экрана более 300 лк.

В качестве источников света в случае искусственного освещения должны применяться преимущественно люминесцентные лампы типа ЛБ. Допускается применение ламп накаливания в светильниках местного освещения.

# 4. ЭКСПЕРИМЕНТАЛЬНЫЙ РАЗДЕЛ

## 4.1. **Тестирование игры**

Игровое приложение содержит в себе некоторую логику – тестирование данной логики есть неотъемлемая часть разработки. Зачастую данный этап может быть более сложен, чем программирование: необходимо учесть все возможные прецеденты развития ситуаций.

Как правило, игровое приложение сначала тестируется самими разработчиками, а затем и пользователями, добровольно решившие помочь на безвозмездной основе в разработке игры. Меж тем тестирование может проводится и в закрытом виде на заранее подготовленных тест-кейсах – что и было проведено.

Таблица 3. – Тест-кейсы игрового приложения

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Действие** | **Ожидаемый**  **результат** | **Фактический**  **результат** | **Пройден** |
| 1 | Клик на игровом объекте. | Происходит некоторое действие. | Появляется информационная панель с игровой подсказкой. | Да |
| 2 | Ввод корректной команды. | Игровое состояние изменяется посредством введённой команды. | Команда влияет на игровое состояние. | Да |
| 3 | Ввод некорректной команды. | Команда не должна затрагивать (изменять) игровое состояние. | Команда не затрагивает игровое состояние, появляется ошибка в окне вывода для разработчика. | Нет |
| 4 | Нажатие кнопки перемещения в начало игры. | Персонаж перемещается в стартовую точку. | Персонаж перемещен в стартовую точку. | Да |
| 5 | Намеренный влет игрового персонажа в игровой объект, не предназначенный для влета. | Диагностирование некорректной ситуации, ее исправление. | Персонаж выводит сообщение о недопустимости такой ситуации, перемещается в стартовую точку. |  |
| 6 | Нажатие кнопки перемещения в меню. | Текущий уровень закрывается, пользователь перемещается в меню. | Текущий уровень закрывается, пользователь перемещается в меню. | Да |
| 7 | Влет в объект, предназначенный для перехода на следующий игровой уровень. | Загрузка следующего уровня. | Следующий уровень корректно загружается, выводя информационную подсказку. | Да |

Данные тесты были составлены и отработаны для понимания, как именно пользователь может взаимодействовать с игрой. Конечно, в процессе тестирования были обнаружены значительные недоработки, которые затем были устранены.

Целесообразным будет разделить тестирование на несколько стадий:

1. альфа-тестирование (первичные результаты);
2. бета-тестирование (полноценный тест игры);
3. релиз-тестирование (тест перед выпуском игры).

Однако из-за того, что созданный игровой проект достаточно небольшой, а функциональность в нем покрывается с помощью описанных выше тест-кейсов – то нет проблем объединить все этапы тестирования в один. Это не является плохой практикой – наоборот, это позволяет убыстрить выход игры, а все найденные недочеты устранять по мере их нахождения.

В целях оптимизации в дальнейшем следует написать юнит-тесты. Эт специальный вид тестирования, при котором тестируется отдельные функциональные элементы игры. Это позволит сократить время разработки – данные тесты автоматизированы. Все, что необходимо делать в процессе разработки – запускать данные тесты ,и наблюдать, что старая функциональность не было сломана, а новая добавлена в таком виде, что тоже покрывается юнит-тестами.

Приоритетным развитием также следует считать использование End-To-End тестов, которые предназначены для тестирования действий реального пользователя.

# ГРАФИЧЕСКИЙ РАЗДЕЛ

В данном разделе рассмотрим наиболее часто используемые изображения в созданной видеоигре. данные спрайты были созданы вручную, то есть весь процесс отрисовки и корректировки был проведен посредством автора проекта.

Отметим, что в банке изображений, к примеру, ShutterStock, можно найти самые разные картинки для дальнейшего встраивания их в игре. Это можно сделать и в официальном репозитории от Unity: там доступна целая плеяда заранее подготовленных моделей, готовые композиции для заданной тематики. Недостаток такого подхода очевиден: за данные изображения нужно платить. Автор проекта не считает целесообразным тратить бюджет игрового проекта на данный пункт, и потому счел необходимым создать вручную все игровые изображения.

Все игровые изображения доступны в отдельной папке игрового проекта под названием Sprites.

Первым изображением, с которого начиналась игра – стал задний фон игры. С него стало понятно, какой жанр следует выбрать, в каком ключе продолжать развитие игры.

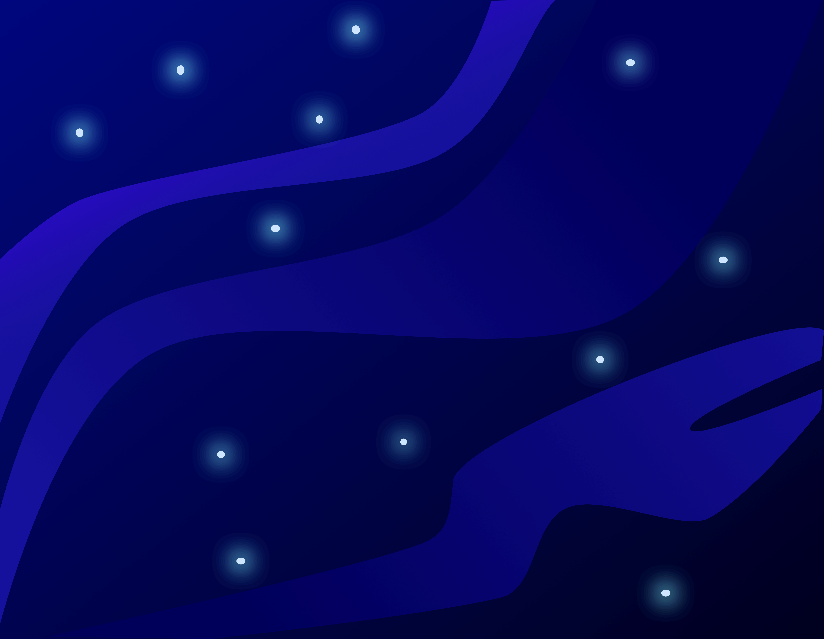


Рисунок 35 – задний фон игры

Следующим шагом стало создание игрового персонажа. Так как игровой персонаж должен быть анимирован – то следовало создать не один отдельный спрайт, а целую группу спрайтов. Данный подход называется созданием тайлсета – создается одно большое изображение, на котором покадрово изображена анимация объекта. В процессе игры вырезается один кадр из данного тайлсета, который затем отображается на экране. Это позволяет убыстрить процесс рисования художнику, а также оптимизировать производительность игры.

При создании игровых спрайтов следует понимать, что чем больше пикселей в изображении – тем тяжелее игровому движку отображать его. Поэтому после нарисования спрайтов изображения были пережаты более чем на 70% - это позволило сократить использование как оперативной, так и статичной памяти в игре.



Рисунок 36 - базовая версия игрового персонажа

После создания тайлсета Unity позволяет вручную определить, какие именно кадры следует использовать. Для этого используется специальный вспомогательный инструмент SpriteEditor: он становится доступен после импортирования изображения в качестве ассета.

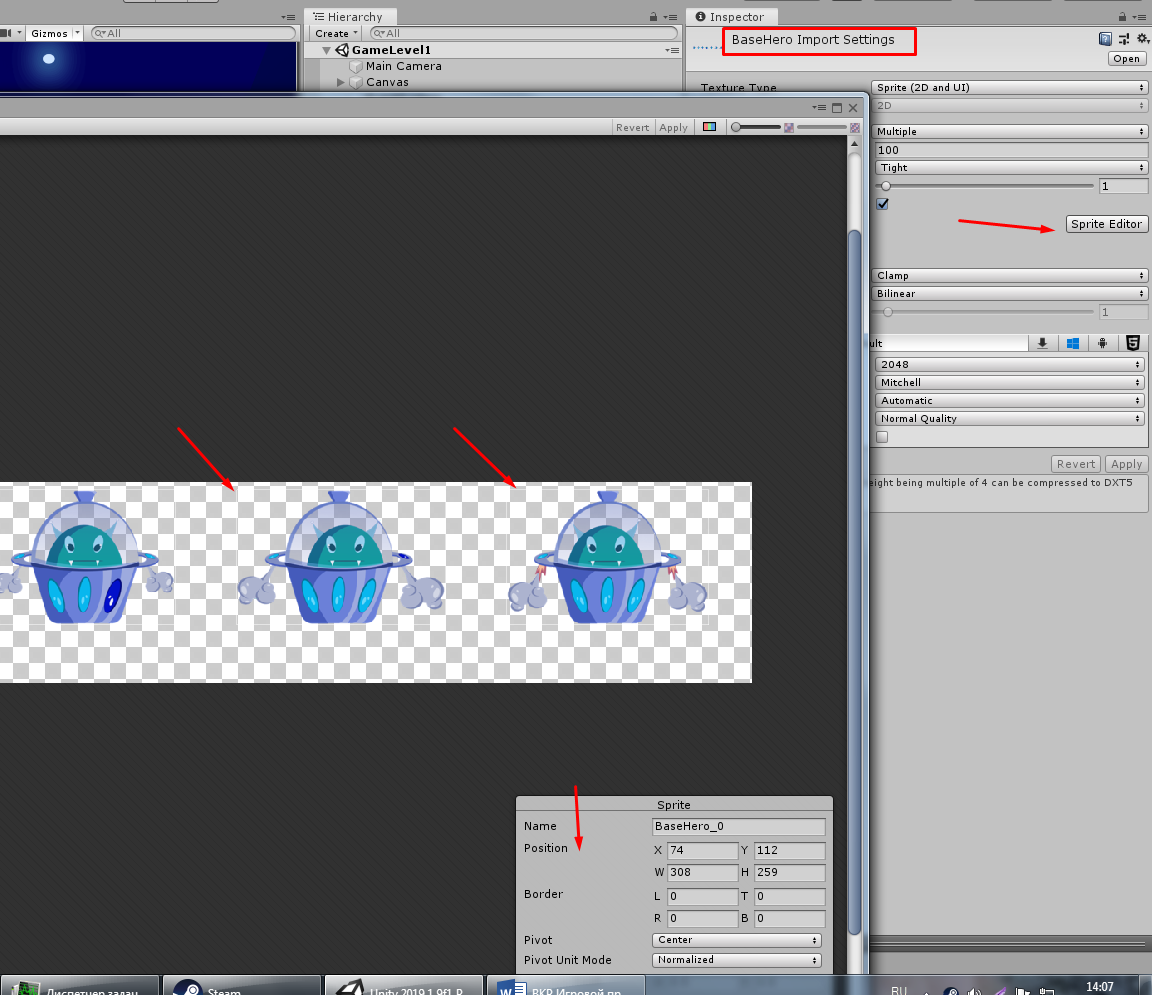


Рисунок 37 - редактор спрайтов

После разбития изображения на отдельные спрайты – каждый таковой спрайт стал доступен в виде отдельного изображения в среде разработки (меж тем все эти спрайты продолжают оставаться фактически единым изображением):



Рисунок 38 - группа спрайтов в тайлсете

Несмотря на то, что игровой персонаж был нарисован – его отображение было связано со следующими проблемами:

персонаж имел разную размерность, поэтому то увеличивался, то уменьшался,

кадры из тайлсета были вырезаны криво, поэтому персонаж дергано перемещался из стороны в сторону.

Было принято решение перерисовать игрового персонажа, добавив большую плавность. А затем в автоматическом режиме разрезать спрайты на одинаковые куски: для этого следовало отцентрировать каждый кадр по заранее определенной области. Благодаря этому удалось добиться плавного отображения персонажа в частоте 60 кадров в секунду, что является высоким стандартом для игровых устройств.

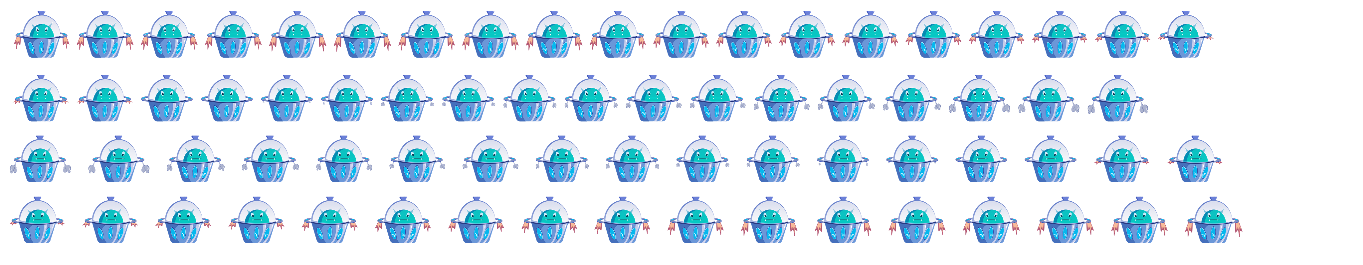


Рисунок 39 - новая версия игрового персонажа

Было подготовлено главное: задний фон игры и ее главный персонаж. Оставалось создать внутри игровое окружение, позволяющее игроку понять, что он погружен в атмосферу и предоставляемый сеттинг. Таким окружением стали планеты и их спутники. Учитывая предыдущий опыт, данные изображения были также созданы в виде тайлсета:



Рисунок 40 – планеты и их луны

Учитывая все вышеизложенное, стало возможным все скомпоновать и создать игровую заставку, которую пользователь видит при запуске новой игры:



Рисунок 41 - игровая заставка

Каждая игра обладает игровым меню. Создаваемая игра не исключение:



Рисунок 42 - игровое меню

Для создания игрового меню использовались изображения в следующем виде:



Рисунок 43 - игровая кнопка

Однако затем стало понятно, что на кнопках нужно будет помещать самый различный текст, поэтому была создана кнопка следующего универсального вида:

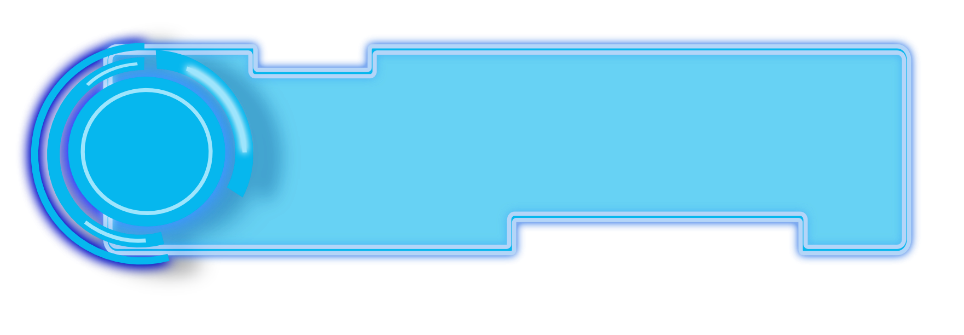


Рисунок 44 - универсальная игровая кнопка

Кнопки в игре должны занимать как можно меньше в процессе игры, а вот до старта игры – наоборот, как можно больше места. Поэтому созданные большие кнопки были удобны только лишь на одной стадии игры – начальной. Дальше следовало переосмыслить вид кнопок, что и было сделано:

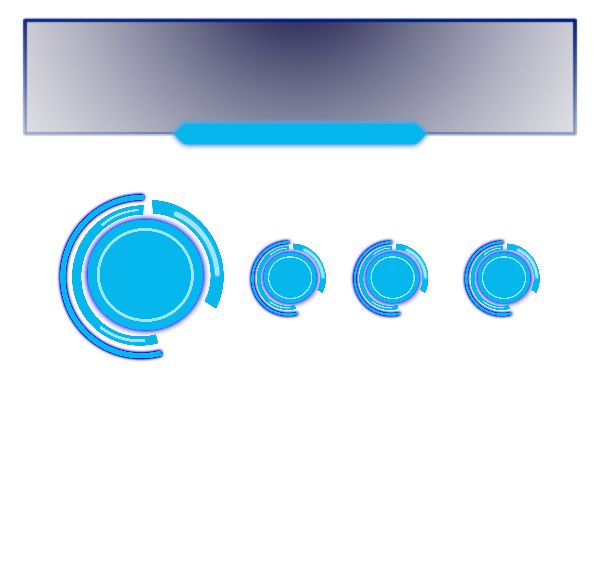


Рисунок 45 – компактные игровые кнопки

Финальным аккордом в отрисовке изображений стало изображение черной дыры. Как и многие игровые объекты – на холсте изображение смотрелось органично, а вот в самой игре было словно не в своей тематике. Так, разработчик понял – необходимо соблюдать стилистику, общий вид игры, чтобы ничего не выбивалось за пределы инактивного восприятия. Приложим исходную и переосмысленную версию черной дыры (переход на следующий уровень):

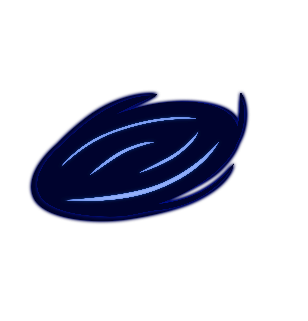


Рисунок 46 - изображение черной дыры

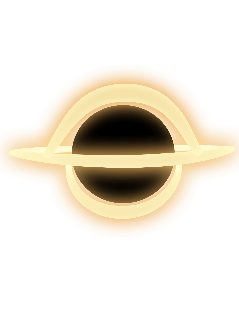


Рисунок 47 - новое изображение черной дыры

Все игровые изображены представлены в данном разделе и являются исходными для игрового проекта.

# ЗАКЛЮЧЕНИЕ

В процессе написания игрового проекта были достигнуты следующие цели:

* изучен контекст разработки видеоигр;
* освоены среды разработки для рисования спрайтов, написания игровой логики;
* изучена работа с анимацией, ее возможности в среде Unity;
* изучены возможности языка C#;
* построен игровой мир со своим собственным сюжетом.

Несомненно, достигнутые цели – это только часть на общем пути игрового разработчика. Отметив данные пункты, прежде всего хотелось бы подтвердить: да, действительно, была создана игра, которую можно демонстрировать рынку видеоигр. И в процессе разработки этой видеоигры были заложены такие фундаментальные навыки, позволяющие продолжить карьеру разработчика.

Несомненно, есть моменты, которые можно улучшить, которые можно оптимизировать – это очевидный на дальнейший задел на будущее, на развитие проекта.

В современном мире существует множество разных возможностей для построения видеоигра. Почему же был выбран Unity? Потому что имено он позволяет создать полноценную игру, не имея каких-либо специфических навыков. Поддержка данной среды продолжается, начиная с самого ее основания. Владелец данного движка – Microsoft, что говорит о стабильности, качестве и оптимистичных планах.

Нельзя не отметить язык разработки – C#. Он, как и Unity, достаточно прост в изучении. Но не в этом его главное преимущество – данный язык активно востребован на рынке труда. Действительно, ведь низкий порог вхождения, оптимизированность, множество разных синтаксических конструкций в языке позволяют стать данному языку уверенным кандидатом на звание одного из самых востребованных языков.

Итак, при построении игрового проекта – был построен не сколько сам проект, сколько были развиты навыки, сколько были получены возможности для дальнейшего уверенного, качественного, позитивного развития разработчика.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Пасынки Вселенной: [Научная фантастика] / Хайнлайн Роберт. – Astounding Science Fiction, 1941. – 256 с. ISBN 978-5-699-19553-4
2. Фленов, Михаил Библия C# / Михаил Фленов. - М.: БХВ-Петербург, 2011. - 560 c.
3. [Электронный ресурс]. – Режим доступа: <https://unity.com/ru/learn> , свободный.
4. [Электронный ресурс]. – Режим доступа: <https://docs.unity3d.com/ru/2018.4/Manual/UnityManual.html> , свободный.
5. [Электронный ресурс]. – Режим доступа: <https://affinity.serif.com/ru/> , свободный.
6. [Электронный ресурс]. – Режим доступа: <https://visualstudio.microsoft.com/ru/msdn-platforms/> , свободный.
7. [Электронный ресурс]. – Режим доступа: <https://metanit.com/sharp/> , свободный.
8. [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/post/110999/> , свободный.
9. [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/post/67431/> , свободный.
10. [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/post/141270/> , свободный.

# ПРИЛОЖЕНИЯ

**Основные понятия**

**FPS как технический показатель в играх** — количество кадров в секунду выводимое на экран. Очень важный показатель для игр в жанрешутера. Количество кадров для комфортной игры начинается от 120 кадров в секуду.

**Game Dev** — Разрабо́тка компьютерных игр — процесс создания компьютерных игр (видеоигр).

**Nodes** — это такие объекты, как события, вызовы функций, операции управления потоком, переменные и т. д.

**Дизайн-документ** — это детальное описание разрабатываемой компьютерной игры.

**Префаб** – подготовленная модель с некоторыми дополнительными компонентами, применяется для многократного переиспользования.

**Компилятор** – программа, позволяющая транслировать код на выбранном программном языке в машинные инструкции.

**Интерактивный объект** – объект, с которым возможно взаимодействие посредством стандартных элементов ввода (мышка / клавиатура / джойстик / тачскрин).

**Скрипт** – программная инструкция, определяющая внутриигровую логику.

**Игровой движок** – совокупность программных средств для обеспечения некоторого функционала для разработки игровых приложений.

**Переменная** – именованная область память, с которой возможно взаимодействие в программном коде.

**Тип переменной** – переменная с типом, предусмотренном в данном программном языке. Тип определяет количество допустимой выделяемой памяти под переменную.

**Цикл** – упорядоченные команды, воспроизводимые до тех пор, пока срабатывает некоторое условие.

**DLL** – динамическая библиотека, предназначенная для встраивания и использования в других программах.

**Спрайт** – игровое изображение, используется в 2D играх.

**Ассет** – импортированный в среду Unity объект, после импорта генерируется специальная мета-информация для среды разработки.

**Тайлсет** – группа спрайтов в одном изображении, предназначенная для раскадровки и последующей анимации.

**Рендерер** – процесс отображения игрового объекта на экране.

**Коллайдер** – модуль, имитирующий физическое столкновение объектов.

**Твердое тело** – модуль, имитирующий физическое тело объектов.